

# Apprentissage pour l'image

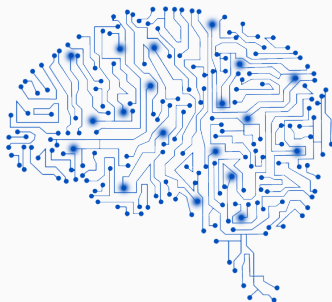
## Machine learning for image processing

### Final slides

---

Emile Pierret

Jeudi 30 mars 2023

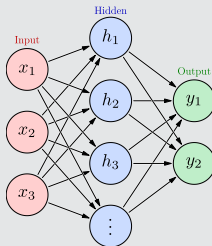


- dataset = ensemble de données
- label = étiquette
- training dataset = ensemble de données d'entraînement
- test dataset = ensemble de données de test
- loss = coût ( à minimiser )
- network = réseau
- Convolutional Neural Network (CNN) = réseau de neurones convolutif
- layer = couche
- hidden layer = couche cachée
- input = entrée
- output = sortie

## Universal Approximation Theorem

(Hornik et al, 1989; Cybenko, 1989)

Toute fonction  $f : K \subset \mathbb{R}^N \rightarrow \mathbb{R}^K$  peut être approximée uniformément par un réseau avec une seule couche cachée avec un nombre suffisant de neurones.



$$\mathbf{h} = g(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{y} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

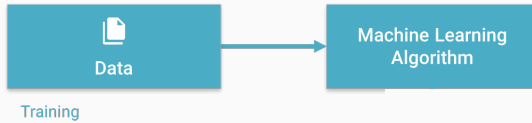
- Le théorème ne renseigne pas sur l'ordre de grandeur du nombre de neurones nécessaire.
- Il n'y a pas de garantie qu'on puisse concrètement optimiser un tel réseau pour toute fonction.

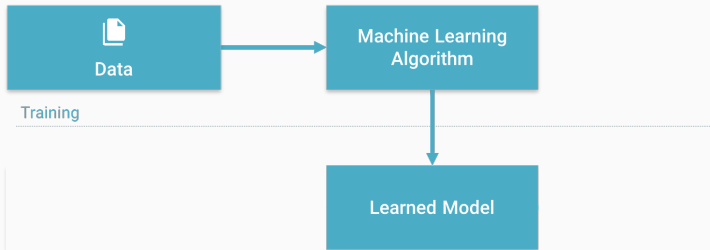
- Nous souhaitons construire des réseaux de neurones pour la classification
- dans un cadre d'apprentissage supervisé

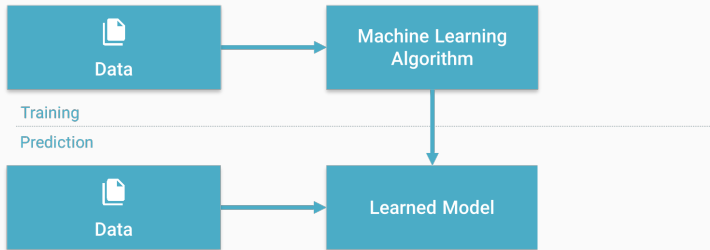
### **Définition (apprentissage supervisé)**

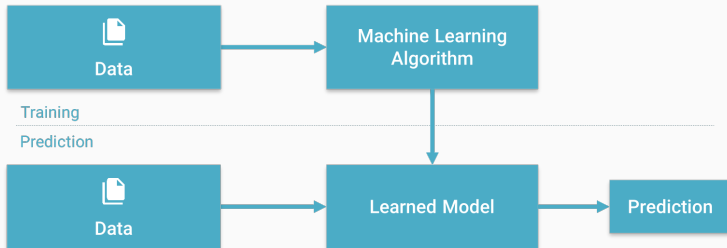
L'apprentissage supervisé consiste à superviser l'apprentissage de la machine en lui montrant des exemples (des données) de la tâche qu'elle doit réaliser.

Dans notre cas, on possède des images déjà étiquetées dans différentes classes.











- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

Traitement de l'ensemble des données :

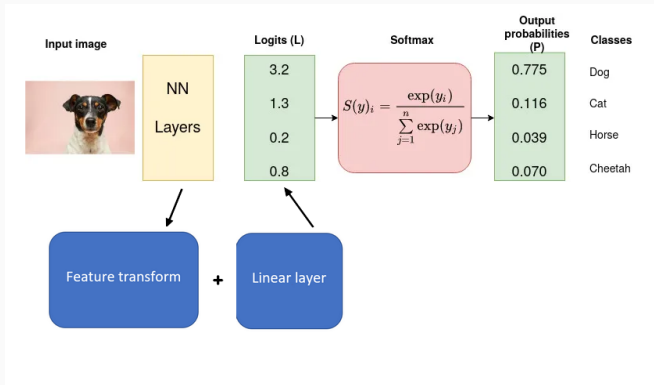
⇒ : Vu aujourd'hui en TP.

⇒ : Nous allons utiliser des ensemble de données déjà implémentés en Pytorch.

⇒ : Déjà implémenté ? Concrètement, on aura un ensemble d'images divisé en 10 classes avec chaque image attachée à une classe (via le label).

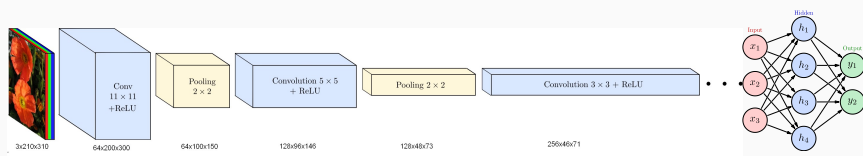
- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

# Comment répondre à une tâche de classification ?





# Pour les images: la feature transform via un CNN



- Une partie convolutive pour l'extraction de features.
- Une partie linéaire pour la classification des features.

Pourquoi un CNN ?

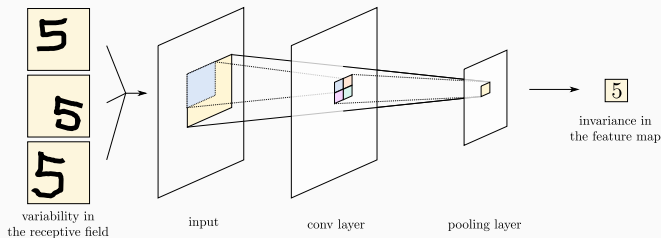
Pourquoi un CNN ?

Un CNN limite le nombre de paramètres en travaillant localement sur les images

# La motivation des CNN

Pourquoi un CNN ?

Un CNN limite le nombre de paramètres en travaillant localement sur les images



- La sortie est **inchangée** même avec des petits changements

Une source qui en parle : ce site (mot avec url)

# Comment coder une architecture réseau ?

```
1 class Net(nn.Module):
2     def __init__(self):
3         super(Net, self).__init__()
4         self.conv1 = nn.Conv2d(3, 64, 11)
5         self.conv2 = nn.Conv2d(64, 128, 5)
6         self.conv3 = nn.Conv2d(128, 256, 3)
7         self.fc1 = nn.Linear(____, 120) # TODO
8         self.fc3 = nn.Linear(120, 10)
9
10    def forward(self, x):
11        # conv1 + Relu + pooling
12        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
13        # conv2 + Relu + pooling
14        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
15        # conv3 + Relu
16        x = F.relu(self.conv3(x))
17        # Vers les couches lineaires
18        x = torch.flatten(x, 1)
19        # 1ere couche lineaire + Relu
20        x = F.relu(self.fc1(x))
21        # 2de couche lineaire
22        x = self.fc2(x)
23        return x
24 net = Net()
```

Pourquoi fc ? Les couches linéaires sont parfois appelées "fully connected".

# Comment coder une architecture réseau ?

```
1 class Net(nn.Module):
2     def __init__(self):
3         super(Net, self).__init__()
4         self.conv1 = nn.Conv2d(3, 64, 11)
5         self.conv2 = nn.Conv2d(64, 128, 5)
6         self.conv3 = nn.Conv2d(128, 256, 3)
7         self.fc1 = nn.Linear(256*46*71, 120) # TODO
8         self.fc2 = nn.Linear(120, 10)
9
10    def forward(self, x):
11        # conv1 + Relu + pooling
12        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
13        # conv2 + Relu + pooling
14        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
15        # conv3 + Relu
16        x = F.relu(self.conv3(x))
17        # Vers les couches lineaires
18        x = torch.flatten(x, 1)
19        # 1ere couche lineaire + Relu
20        x = F.relu(self.fc1(x))
21        # 2de couche lineaire
22        x = self.fc2(x)
23        return x
24 net = Net()
```

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

## Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?



## Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant  $\mathcal{E}$  l'ensemble d'entraînement, le coût  $L$  s'écrit sous la forme :

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

(Voir les slides Presentation\_TP1)

## Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant  $\mathcal{E}$  l'ensemble d'entraînement, le coût  $L$  s'écrit sous la forme :

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

(Voir les slides Presentation\_TP1)

- Et en Python ?

# Comment choisir le loss ?

- Nous avons construit une fonction de coût, une loss, nommée Cross-entropy
- D'où vient-elle ?
- La maximisation de la log-vraisemblance avec l'hypothèses que les éléments de l'ensemble des données d'entraînement sont indépendantes.
- En notant  $\mathcal{E}$  l'ensemble d'entraînement, le coût  $L$  s'écrit sous la forme :

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

(Voir les slides Presentation\_TP1)

- Et en Python ?

```
1 criterion = nn.CrossEntropyLoss()
```

On peut choisir d'autres loss:

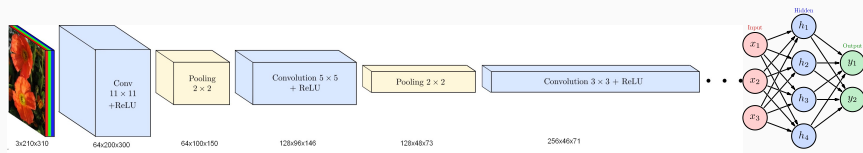
```
1 criterion = nn.MSELoss()
```

```
2 criterion = nn.L1Loss()
```

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement
- Enfin, il y a une phase de test

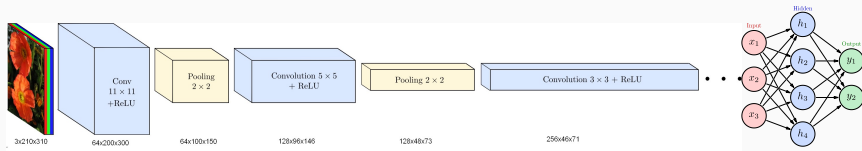
# Qu'est-ce que $W$ ?

Dans le réseau ci-dessous, que chercher-t-on à optimiser ?



# Qu'est-ce que $W$ ?

Dans le réseau ci-dessous, que chercher-t-on à optimiser ?



L'architecture du réseau ne va pas changer au cours de l'entraînement :

- La taille des noyaux
- Le nombre de canaux de sortie
- Les fonctions d'activation
- La taille des matrices de la partie linéaire

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

On découpe l'ensemble de données  $\mathcal{E}$  en "batches"  $\mathcal{B}$  tels que  $\mathcal{E} = \bigcup \mathcal{B}$

## Comment optimiser le réseau ?

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

On découpe l'ensemble de données  $\mathcal{E}$  en "batches"  $\mathcal{B}$  tels que  $\mathcal{E} = \bigcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch  $\mathcal{B}$  de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer  $\ell(W, \mathbf{x})$  pour  $\mathbf{x}$  dans le batch
    - Calculer  $\mathbf{g} = \sum_{\mathbf{x} \in \mathcal{B}} \nabla_{\mathbf{W}} \ell(W, \mathbf{x})$
    - Faire un pas de SGD :

$$\mathbf{W} \leftarrow \mathbf{W} - \gamma \mathbf{g}$$

---



## Comment optimiser le réseau ?

$$L(\mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{E}} \ell(\mathbf{x}, \mathbf{W})$$

On découpe l'ensemble de données  $\mathcal{E}$  en "batches"  $\mathcal{B}$  tels que  $\mathcal{E} = \bigcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch  $\mathcal{B}$  de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer  $\ell(W, \mathbf{x})$  pour  $\mathbf{x}$  dans le batch
    - Calculer  $\mathbf{g} = \sum_{\mathbf{x} \in \mathcal{B}} \nabla_{\mathbf{W}} \ell(W, \mathbf{x})$
    - Faire un pas de SGD :

$$\mathbf{W} \leftarrow \mathbf{W} - \gamma \mathbf{g}$$

- 
- époque = désigne un passage complet du jeu de données d'entraînement par l'algorithme.
  - batch = morceau du jeu de données
  - Pour cette raison, input d'un réseau de la forme  $[b, c, M, N]$
  - Comment est calculé le gradient ?

## Un paramètre supplémentaire : le moment (momentum)

- Pour chaque époque :
  - Pour chaque batch  $\mathcal{B}$  de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer  $\ell(W, \mathbf{x})$  pour  $\mathbf{x}$  dans le batch
    - Calculer  $\mathbf{g} = \beta \mathbf{g} + \sum_{\mathbf{x} \in \mathcal{B}} \nabla_{\mathbf{W}} \ell(W, \mathbf{x})$
    - Faire un pas de SGD :

$$\mathbf{W} \leftarrow \mathbf{W} - \gamma \mathbf{g}$$

→ C'est une façon d'accélérer la convergence

## Comment optimiser le réseau avec Python ?

```
1 optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
2 for epoch in range(Nb_epochs):
3     for i, data in enumerate(trainloader, 0): #Divise l'ensemble de
        donnees en batches
4
5         batch, labels = data
6
7         # zero the parameter gradients
8         optimizer.zero_grad()
9
10        # forward + backward + optimize
11        outputs = net(inputs)
12        loss = criterion(outputs, labels)
13        loss.backward()
14        optimizer.step()
```

- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement ✓
- Enfin, il y a une phase de test

## Comment vérifier la performance du réseau ?

- En utilisant l'ensemble de données de test.

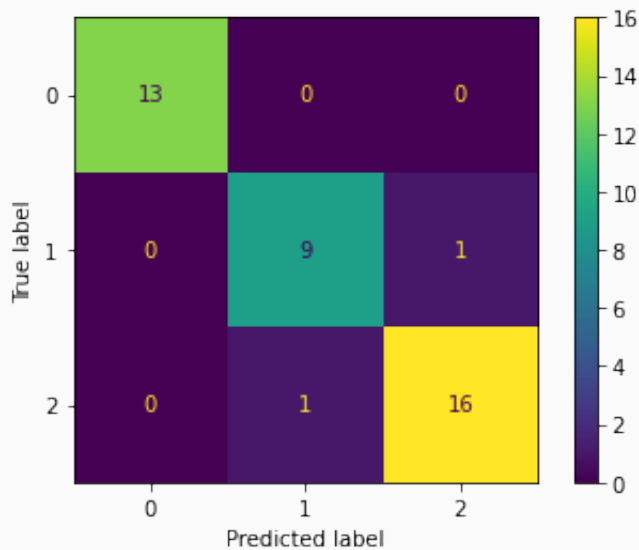
```
1 from sklearn.metrics import classification_report
2 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
3 #Classification report
4 print(classification_report(t_test,pred))
5
6 #Confusion matrix
7 print(confusion_matrix(t_test,pred))
8 disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(
    t_test,pred))
9 disp.plot()
```

# Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.90	0.90	0.90	10
2	0.94	0.94	0.94	17
accuracy			0.95	40
macro avg	0.95	0.95	0.95	40
weighted avg	0.95	0.95	0.95	40

Les formules utilisées pour construire le tableau peuvent être trouvées à ce lien

## Matrice de confusion



- Pour une tâche de classification, nous avons besoin :
  - d'un ensemble de données divisé en test et entraînement ✓
  - d'une architecture de réseau ✓
  - d'un coût à minimiser ✓
- Ensuite, il y a une phase d'entraînement ✓
- Enfin, il y a une phase de test ✓



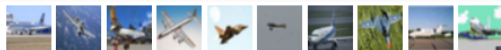
Les problèmes que nous ne nous sommes pas posés :

- Choix de l'architecture du réseau  
→ décidée avec un ensemble de validation et l'état de l'art
- Choix des hyperparamètres d'optimisation ( $\text{lr}$ , momentum)  
→ décidée avec l'affichage du coût au cours du temps
- Ajout de couche "batchnorm" pour améliorer l'entraînement  
→ traité en M2 avec B. Galerne

# En TP aujourd'hui:

Entraînement d'un réseau pour classifieur CIFAR10.

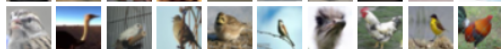
**airplane**



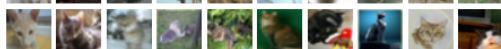
**automobile**



**bird**



**cat**



**deer**



**dog**



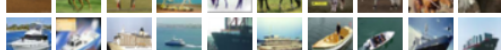
**frog**



**horse**



**ship**



**truck**



+ retour sur le TP1

## En projet:

Entraînement d'un réseau pour classifier MNIST.



Choix de la date de rendu du projet ?

- Votre semaine d'examen : du 17 au 21 avril.
- Votre départ en stage : 6 mai ?

**Questions?**

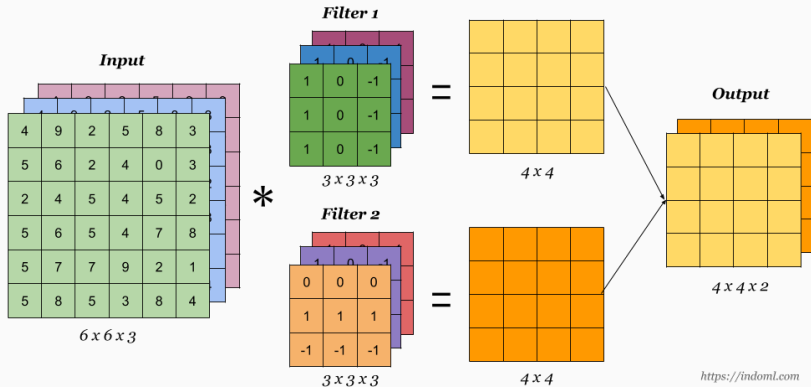
1	1	1	0	0
0	1	1	1	0
0	0	1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>
0	0	1 <sub>x0</sub>	1 <sub>x1</sub>	0 <sub>x0</sub>
0	1	1 <sub>x1</sub>	0 <sub>x0</sub>	0 <sub>x1</sub>

Image

4	3	4
2	4	3
2	3	4

Convolved  
Feature

# Convolution avec différents canaux (sans biais)



# Pooling

